Experience in spacecraft on-board software development

Juan A. de la Puente, Alejandro Alonso, Juan Zamorano, Jorge Garrido, Emilio Salazar, Miguel A. de Miguel

Universidad Politécnica de Madrid, ETSIT, Avda. Complutense 30, E-28040 Madrid, Spain

Abstract

This paper describes some important aspects of high-integrity software development based on the authors' work. Current group research is oriented towards mixed-criticality partitioned systems, development tools, real-time kernels, and language features. The UPMSat-2 satellite software is being used as technology demonstrator and a case study for the assessment of the research results. The flight software that will run on the satellite is based on proven technology, such as GNAT/ORK+ and LEON3. There is an experimental version that is being built using a partitioned approach, aiming at assessing a toolset targeting partitioned multi-core embedded systems. The singularities of both approaches are discussed, as well as some of the tools that are being used for developing the software.

Keywords: Real-time systems, model-driven engineering, Ada.

1 Introduction

The UPM STRAST group has a long time experience in developing high-integrity real-time systems. The group research in this domain is currently oriented towards mixed-criticality partitioned systems, development tools, real-time kernels, and language features. In order to validate technical achievements in this field, the UPMSat-2 satellite software is being used as a case study. In this paper, ongoing work and experiences from this development are described.

UPMSat-2 is a project aimed at building a micro-satellite that can be used as a platform for experimenting with various technologies and acquiring long-term experience in different aspects of space systems. The project is being carried out by a multi-disciplinary team at UPM, with the collaboration of several research groups and industrial companies. The satellite is expected to be launched in the final quarter of 2015. STRAST is responsible for developing all the software required for the mission, including on-board software for platform and payload management. The flight software is built as a monolithic system, running on top of an ORK+ kernel on a LEON3 [**?**] computer board. The software is being developed according to the provisions in the ECCS-E-ST-40 [**?**] and ECCS-Q-ST-80 [**?**] standards, in order to ensure that the final software product can be validated for the mission.

Mixed-criticality systems are raising a growing interest in the area of embedded systems, due to their potential for improving software productivity and quality. In the context of the MultiPARTES and HI-PARTES projects, methods and tools for mixed-criticality partitioned multi-core embedded systems are being developed. One of the responsibilities of the group is the development of a toolset for supporting this approach. In this context, UPMSat-2 is being used as a case study. In particular, a partitioned implementation running on a XtratuM hypervisor [**?**] is being developed for demonstration and validation of the project outcomes.

The methodological and architectural approaches used in this work is described in the rest of the paper. Section **??** contains an overview of the satellite system and the architecture of the on-board computer. The main software subsystems and the architectural approaches are discussed also described in this section. Section **??** describes the development tools used. Some details of the validation facility are presented in section **??**. Finally, a summary of the lessons learned so far and plans for the next future is presented in section **??**.

2 The UPMSat2 On-Board Software System

2.1 Overview of the satellite system

UPMSat-2 is a micro-satellite with a geometric envelope of $0.5 \times 0.5 \times 0.6\,\mathrm{m}$ and an approximate mass of $50\,\mathrm{kg}$ (figure **??**). It will describe a low Earth noon sun-synchronous polar orbit [**?**] with a period about $97\,\mathrm{min}$. There are two visibility periods from the ground station every 24 hours, with an approximate duration of $10\,\mathrm{min}$ each.

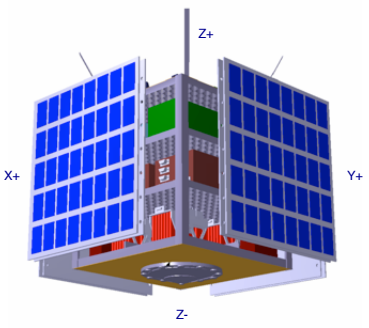


Figure 1: General view of the satellite platform.

Electrical power is provided by solar panels and batteries. Voltage control is analog, keeping voltages for the satellite subsystems within appropriate ranges.

The attitude of the satellite is computer-controlled, using basic sensors and actuators. The attitude is determined by means of magnetometers, which provide a measurement of the Earth magnetic field vector in the satellite reference frame. Deviations are corrected by means of magnetorquers, which create a magnetic field that makes the satellite rotate accordingly.

Communications with the ground station are carried out by means of a dual radio link in the VHF 400 MHz band, with a raw transfer rate of 9600 bit/s. A simplified version of the X.25 data link layer protocol is used for error control and packet transmission.

The payload of the satellite consists of a set of experiments focused on testing different kinds of equipment in a space environment. The experiments have been proposed by industry and some research groups.

There is a single on-board computer (OBC) that executes all the data handling, attitude control, and telecommunications functions. It is based on a LEON3 processor implemented on a radiation-hardened FPGA, with 4 MB RAM, 1 MB EEPROM, and digital and analog interfaces. The on-board software system runs on this hardware platform.

2.2 Software functionality

The main functions of the on-board software can be grouped as follows:

- Platform monitoring and control *(housekeeping)*. Platform data, such as voltages and temperatures at different points, are periodically sampled and checked in order to assess the status of the satellite.
- On-board data handling (OBDH), including decoding and executing telecommands (TC) received from the ground station, and composing and sending telemetry (TM) messages with housekeeping data, event and error logs, or experiment results.
- Attitude determination and control (ADCS). Magnetometer values are read periodically, and used by the control algorithm to compute the intensity output to the magnetorquers in each sampling period.

  Alternative ADCS devices, such as solar sensors or reaction wheel actuators, as well as variations in the control algorithm, will be tested as experiments.
- Experiment management. Most of the experiments require control actions to be executed on them, and sensor data to be collected and sent to ground to be analysed.

Figure **??** shows the software context and the top-level functional blocks.

A key concept in on-board software systems is that of operating modes. The system may be in different modes, and may perform different functions, or execute them in different ways, according to the operating conditions of the system.

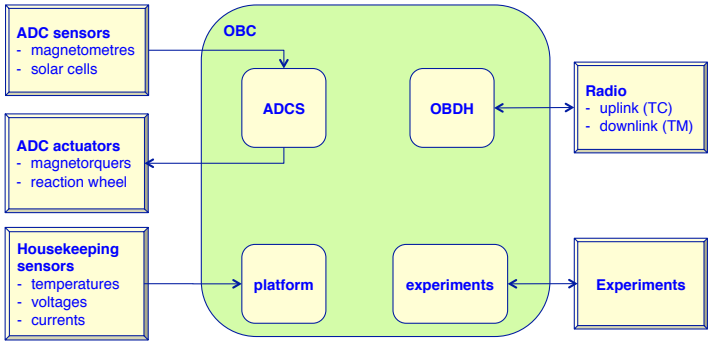


Figure 2: Context and top-level functions.

Figure **??** shows the main operating modes of the on-board software and the events that trigger mode transitions.

The specific functions that are executed in each mode are:

- Initialization mode: load executable code, start execution, and configure I/O devices.
- Nominal mode: housekeeping, OBDH and ADCS as above defined.
- Safe mode: same as nominal mode, with longer periods and reduced functionality in order to save energy power.
- Latency mode: the computer is switched off until batteries are charged (signalled by a hardware timer).
- Experiment mode: one of the experiments is executed, with changes to nominal behaviour if required by the experiment.

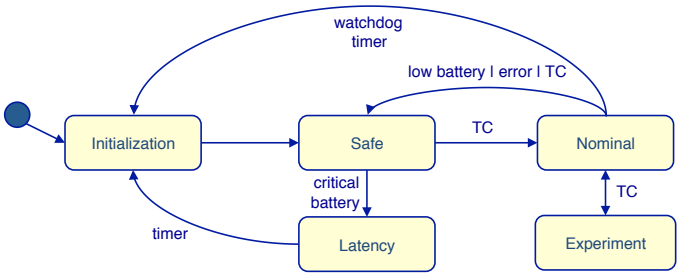


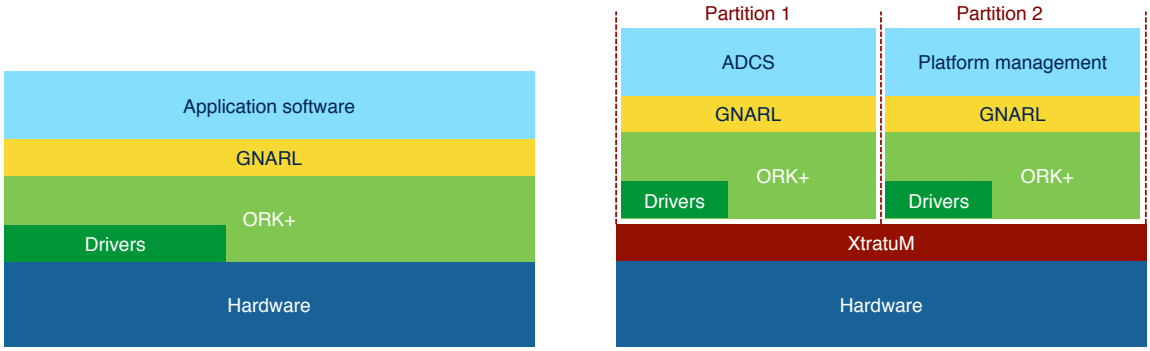
Figure 3: Satellite operating modes.

2.3 Architectural approaches

In order to ensure a timely implementation of the flight software, a monolithic implementation has been designed, using a well known architecture based on GNAT/ORK (figure **??**).

On the other hand, there is growing interest on developing satellite software on partitioned architectures, as exemplified by recent work directed by ESA/ESTEC to develop a partitioned version of the EagleEye reference mission software [**?**].

The MULTIPARTES project [**?**] is aimed at developing tools and solutions based on mixed criticality virtualization for multicore platforms. The virtualization kernel is based on XtratuM, a cost-effective open source hypervisor specifically developed for real-time embedded systems [**?**]. The UPMSat-2 software is being used in MultiPARTES as a case study for



(a) Monolithic architecture. **(b) Partitioned architecture.**

Figure 4: Software architecture.

validating the mixed-criticality technology developed in the project. To this purpose, a partitioned version of the software system is being developed. The partitions run on an adapted version of GNAT/ORK for XtratuM [?]. An example showing the ADCS control subsystem running in one partition and the platform manager providing access to devices in another is shown in figure **??**.

3 Development tools

The increasing complexity of high integrity embedded systems and the need to comply with demanding safety-related standards require suitable toolsets for supporting developers. Model Driven Engineering (MDE) is an appropriate software development approach, that enables the abstraction level of languages and tools used in the development process to be raised. It also helps designers to isolate the information and processing logic from implementation and platform aspects. A basic objective of MDE is to put the model concept on the critical path of software development. This notion changes the previous situation, turning the role of models from contemplative to productive.

The STRAST group has been working with this technology for a long time. The ASSERT project[1] explored the use of MDE technology in space software systems, from which different sets of tools emerged. One of them evolved under the auspices of ESA, resulting in the TASTE toolset [**?**]. TASTE supports a wide set of modelling languages, such as Simulink [**?**] and SDL [**?**], and uses AADL [**?**] as a glue for architecture modelling. The TASTE tools generate Ravenscar Ada code that can be compiled with GNAT/ORK, and are being used as the primary toolset for the monolithic implementation of the UPMSat-2 software.

Another follow-up of ASSERT was the CHESS project,[2] which was focused on property preservation and composability. In the context of this project, an MDE framework for high-integrity embedded systems was originally developed [**?**]. In this framework, the functional part of the system is modelled using UML [**?**]. Models can be enriched with non-functional annotations, in order to integrate different aspects of the software in a single model. This approach has a number of advantages, as it makes models maintenance easier, enables efficient communication within the development team, and supports the validation and analysis of models.

The framework relies on the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [**?**] to describe real-time requirements, properties, resource usage information, and other non-functional properties. A response-time analysis model is automatically generated, which can be used to validate real-time requirements. Finally, source code skeletons in Ravenscar Ada are generated for the main system components.

Mixed-criticality systems are emerging as a suitable approach for dealing with system complexity and reducing development costs, by integrating applications with different criticality levels on the same hardware platform. A separation kernel provides isolation mechanisms in order to guarantee that applications do not interfere with each other. In this way, it is possible to certificate or qualify applications with different criticality levels in an independent way.

Partitioned systems are a remarkable way of providing isolation to applications. In these systems, the separation kernel can be built as a hypervisor that implements a number of partitions as virtual machines isolated from each other in the time and space domains. In this way, applications with different criticality levels can run in different partitions without experimenting any interference from other applications. This approach makes the system development more difficult, as its time behaviour may get much more complex, and requires the hypervisor to be carefully configured.

In the context of the MultiPARTES project, the original CHESS framework is being improved in order to deal with mixed-criticality systems [**?**]. The first activity accomplished is the identification of toolset requirements, which are driven by the inputs from industrial applications in domains such as aerospace, automotive, video surveillance or wind power generation. The most relevant requirements are:

- Development of mixed-criticality systems: The concept of criticality should be central in the system.

[1] Automated proof-based System and Software Engineering for Real-Time systems. FP6 IST 004033.

[2] Composition with Guarantees for High-integrity Embedded Software Components Assembly, ARTEMIS-2008-1-100022.

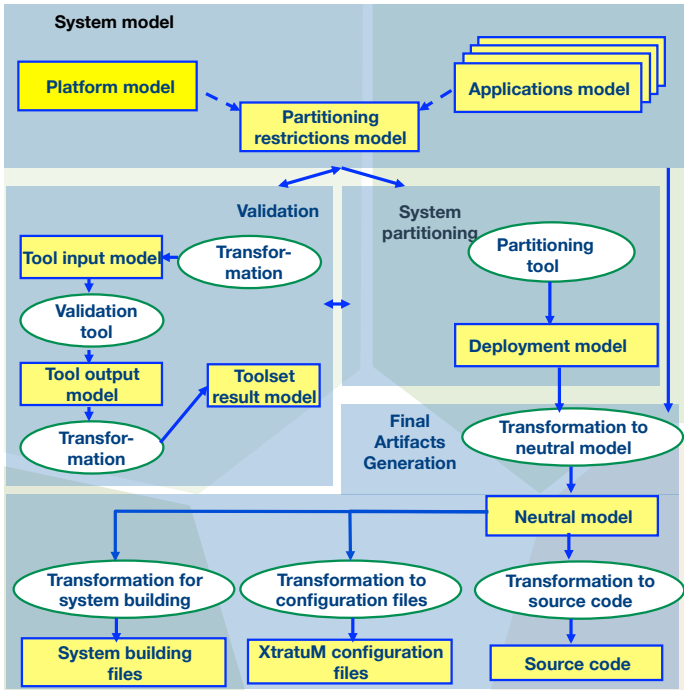


Figure 5: Architecture of the real-time safety systems development framework.

- Support for non-functional requirements: The framework has to provide mechanisms for specifying and validating these requirements. Additionally, the integration of requirements from different components must be supported. Currently safety, real-time, and security requirements are being considered.
- Support for multi-core architectures: Current processor technologies are more and more based on multicores. Design aspects such as modelling, partition allocation, or response time analysis, should be supported.
- System modelling: The toolset must provide means for modelling the whole system, including the applications, platform, and any other elements required for its description.
- Support for system deployment: this implies the generation of a bootable software image with the hypervisor and the partitions code, including the operating systems and applications allocated to them.

The current framework design is shown in figure **??**. A system model is composed of three models:

- Platform model: it describes the execution platform, including hardware, hypervisor, operating system. The platform model relies on UML-MARTE, with some extensions.
- Applications model: it includes the functional model and the required non-functional properties.
- Partitioning restrictions model: it describes the restrictions to be fulfilled by a valid partitioning of the system.

Platform and applications models are independent of a particular system. In this way, it is possible to reuse them in different developments. The restrictions model includes information that applies to a particular system, and any specific criteria for partitioning. Restrictions may include statements that must be fulfilled by a valid partitioning, such as “an application must be allocated to a given partition”, “an application must (not) be in the same partition as another one”, “an application requires a particular hardware device”, or “a partition or application must run on a given core or processor”.

The system partitioning component is in charge of generating a valid system partitioning, i.e. a number of partitions, an allocation of applications to partitions, and an assignment of computational resources to partitions. This information is described in the deployment model. A deployment model must meet the defined restrictions, as well as the specification of non-functional requirements. For example, if an application has a certain criticality level, it must not be allocated to the same partition as a non-critical application.

Some non-functional requirements may be difficult to validate. The validation component can provide validation tools for some of them. For example, a validation tool can support the validation of real-time requirements by carrying out a response time analysis of the system.

As above, inputs to validation processes can be generated automatically. Failure to validate one or more requirements can provide feedback to add restrictions to the partitioning model, so that an alternative deployment model can be produced.

The final step of development consists of the generation of the following final artefacts:

- Hypervisor configuration files, for implementing a behaviour compliant with the deployment model.
- System building files, for automatically generating the final executable system.

- Source code skeletons, for the main entities.

The framework is currently under development. There are working versions of system model, and the artifacts generation tools. With respect to the partitioning component, it is possible to define partitions manually. The automatic partitioning algorithm is under development. Finally, there is an ongoing work on the support for response time analysis of partitioned multi-core systems. The framework is currently tailored for a LEON3 hardware architecture, the XtratuM hypervisor, and the ORK+ operating system. Code skeletons generation is targeted to Ravenscar Ada.

4 Software validation approach

The flight version of the on board computer, based on a LEON3 processor, is still under development. For this reason, an engineering model is currently being used for preliminary software validation. The engineering model is based on a GR-XC3S1500 Spartan3 development board with a LEON2 processor at 40 MHz clock frequency and 64 MB of SDRAM. Cache memory is not used in this implementation. The main difference between the LEON2 processor used in the engineering model and the envisaged production LEON3 are that the latter has a 7-stage pipeline instead of the 5-stage pipeline of LEON2. The differences between these versions of the processors are not significant as they are not central for system behaviour.

The engineering version of the OBC is being used to test and analyse some parts of the software that already mature enough for preliminary validation. For example, there is a working version of the ADCS subsystem, implementing an elaborate attitude control algorithm that has been designed by aerospace engineers, based on a mathematical model of the spacecraft dynamics and the torque perturbations. Due to the complexity of the model, a functional model has been created with Simulink in order to design, test and validate the structure of the control algorithm and to tune its parameters to the most appropriate values.

As it is not possible to test the satellite software in its real environment, a software validation facility (SVF) including hardware-in-the-loop (HIL) simulation has been built. The basic idea is to test the embedded system against a simulation model instead of the real environment. The test environment includes a simulation model that interacts with the control module running on the real computer, as shown in figure **??**. Some additional components have been included as well, in order to model the sensors and actuators that carry out the interaction between the computer and the modelled environment. The tests performed so far show a correct behaviour of the attitude control software. The SVF approach has also been used to analyse the worst-case execution time and the maximum response time of the attitude control procedure [**?**, **?**].

In order to validate the partitioned architecture described in **??**, a prototype has been built with the two partitions shown in figure **??**. The partitioned system runs on the on-board computer. The platform management partition interacts with the SVF computer. The ADCS partition executes the control

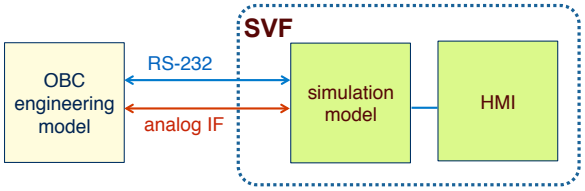


Figure 6: Architecture of the software validation facility.

algorithm, and interacts with the other partition in order to exchange data with the SVF simulation of the spacecraft dynamics.

The results of this exercise have been quite promising. System partitioning has been straightforward using the development framework. The original application was split into two components that were allocated to different partitions. Communication was performed using the XtratuM mechanisms for inter-partition interaction. Partitioning and resource assignment have been done manually. The generation of the artefacts has simplified the deployment of the final system. The next steps include to continue the UPMSat-2 development, and to apply this technology to a more complex system. We are also planning to make an assessment of the response time analysis facilities, in order to guarantee the required timing behaviour.

5 Conclusions and future plans

Model-driven engineering has lived up to its promise to raise abstraction level and make software development easier. Being able to reason with models in the development of the UPMSat-2 software is a real gain over older development methods, and lets the design team concentrate on the system behaviour rather than implementation details.

The use of the TASTE toolset for the monolithic software version is straightforward. We are modelling most of the system behaviour with SDL, except for the ADCS component which is being modelled with Simulink. Automatic code generation from the models has enabled experimenting with the ADCS algorithm and carry out analysis and testing procedures on this subsystem at an early stage. The implementation of this software version on the monolithic GNAT/ORK/LEON3 platform is also straightforward, as this is a well-known and proven platform for space systems.

The partitioned version of the software leaves room for experimenting newer software development methods and tools. Using a partitioned architecture allows the design engineers to separate subsystems from each other and assigning them different criticality levels. For example, the system manager, the ADCS, and the communications (TTC) subsystems can be assigned level B (as per ECSS-Q-ST-80), whereas the experiments in the payload can be considered level C, as they are not critical for the success of the mission. Even subsystems with the same criticality level can profit from partitioning, as they can be validated independently, thus simplifying qualification. However, the virtualization kernel has to be qualified at the maximum criticality level of the partitions, which may in turn make the qualification process more complex. This

issue is being addressed in the MultiPARTES project, where a roadmap to the qualification of the satellite software case study based on UPMSat-2 is being developed.

Effective use of a partitioned approach such as discussed above requires support from a toolset that integrates partitioning with modelling and code generation. The toolset that has been described in the paper has already shown a high potential for this development paradigm, and is also being completed in the framework of MultiPARTES.

Other topics dealt with in the paper, such as the use of a software validation facility with a hardware-in-the-loop configuration and the way to overcome some minor inconsistencies in the design of actuator control tasks, are directly extracted from the authors' experience and have also contributed to clarifying the development process

Plans for the near future include completing the design and implementation of the monolithic software system, and carrying out the complete validation and qualification activities as required by the ESA standards. This requires testing on the flight computer with the actual I/O devices.

With respect to the partitioned software systems, the immediate tasks are completing the toolset, finalise the software design (note that the functional model is the same as in the monolithic version), and complete the roadmap to ESA qualification.

Acknowledgments.

The work described in this paper has been partially funded by the Spanish Government, project HI-PARTES (TIN2011-28567-C03-01), and by the European Commission FP7 programme, project MultiPARTES (IST 287702).

The UPMSat-2 project is led by IDR/UPM.[3] We would like to acknowledge the collaboration of the IDR team, TECNOBIT, as well as the MultiPARTES consortium members.

[3] Instituto Ignacio da Riva, `www.idr.upm.es`.